libORFA — PC-side gate library for Open Robotics Firmware Architecture

Создано системой Doxygen 1.5.9

Wed Aug 12 19:50:59 2009

Оглавление

Глава 1

Список задач

Файл orfa.c Test timeouts

Файл lsorfa.c Add baudrate cli option
Add address cli option

Глава 2

Алфавитный указатель групп

2.1 Группы

Полный список групп.

Глава 3

Дерево директорий

3.1 Алфавитный указатель директорий

Дерево директорий

Глава 4

Алфавитный указатель структур данных

4.1 Структуры данных

Структуры данных с их кратким описанием.

Глава 5

Список файлов

5.1 Файлы

Полный список документированных файлов.

Глава 6

Группы

6.1 Introspection

Структуры данных

- struct orfadrivers

  Driver list.

Функции

- uint8_t orfa_get_drivers_count (struct orfadev *device, int16_t address)
- struct orfadrivers * orfa_get_drivers (struct orfadev *device, int16_t address)
- bool orfa_free_drivers (struct orfadrivers **drivers)

6.1.1 Функции

6.1.1.1 bool orfa_free_drivers (struct orfadrivers ** drivers)

Free drivers list

Аргументы:

← *drivers pointer to drivers list head

Возвращает:

true if success, false if fails

См. определение в файле intro.c строка

6.1.1.2 struct orfadrivers* orfa_get_drivers (struct orfadev * device, int16_t address) [read]

Get drivers list Read drivers list from device. Base command: S <adr+w> 00 01 S <adr+r> <6*count> P

Аргументы:

← *device pointer to device

← address if < 0 then used local address

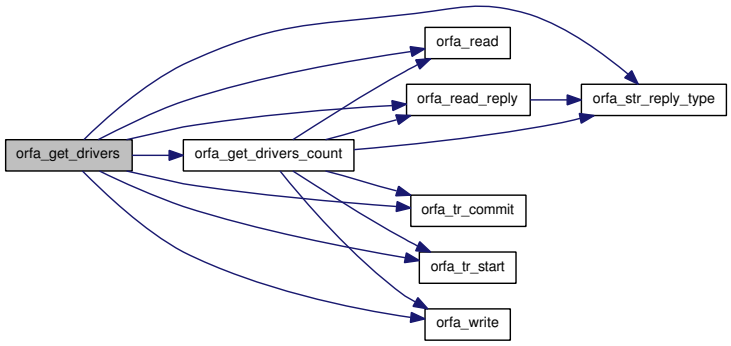
Возвращает:

pointer to drivers list head, NULL if fails

См. определение в файле intro.c строка

Перекрестные ссылки orfareply::data, orfareply::OR_READ, orfareply::OR_WRITE, orfa_get_drivers_count(), orfa_read(), orfa_read_reply(), orfa_str_reply_type(), orfa_tr_commit(), orfa_tr_start(), orfa_write(), orfareply::size, orfareply::type и orfadrivers::uid.

Граф вызовов:



6.1.1.3 uint8_t orfa_get_drivers_count (struct orfadev * device, int16_t address)

Get drivers count Read drivers count from device. Command: S <adr+w> 00 00 S <adr+r> 01 P

Аргументы:

← *device pointer to device

← address if < 0 then used local address

Возвращает:

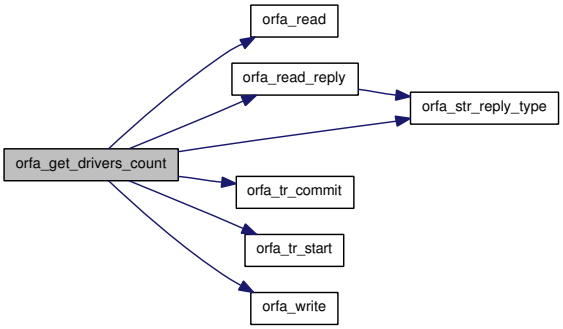
count if success, 0 if fails

См. определение в файле intro.c строка

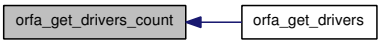
Перекрестные ссылки orfareply::data, orfareply::OR_READ, orfareply::OR_WRITE, orfa_read(), orfa_read_reply(), orfa_str_reply_type(), orfa_tr_commit(), orfa_tr_start(), orfa_write(), orfareply::size и orfareply::type.

Используется в orfa_get_drivers().

Граф вызовов:



Граф вызова функции:



6.2 Device

Структуры данных

- struct orfadev

  Device structure.

- struct orfadev * orfa_open (char *port, int baudrate)
- int orfa_close (struct orfadev **device)

6.2.1 Функции

6.2.1.1 int orfa_close (struct orfadev ** device)

Close device and free memory.

Аргументы:

← **device pointer to device pointer

Возвращает:

close() ret

См. определение в файле orfa.c строка

6.2.1.2 struct orfadev* orfa_open (char * port, int baudrate) [read]

Open device

Аргументы:

← *port serial port device path string (/dev/ttyS0)

← baudrate 115200, 9600

Возвращает:

pointer to device structure or NULL if fails

См. определение в файле orfa.c строка

Перекрестные ссылки orfadev::baudrate, orfadev::clock, orfadev::fd, orfadev::in_transaction, orfadev::local, PCOMMAND, orfadev::port и orfadev::vars.

6.3 Reply

Структуры данных

- struct orfareply

  Reply structure.

Макросы

- #define REPLY_DATA_SIZE 65

  Maximum reply size (ORFA's buffers CBF_SIZE=128, BUF_LEN=65).

- struct orfareply * orfa_read_reply (struct orfadev *device)
- const char * orfa_str_reply_type (struct orfareply *reply)

6.3.1 Функции

6.3.1.1 struct orfareply* orfa_read_reply (struct orfadev * device) [read]

Get request reply Don't forget free() result

Аргументы:

← *device pointer to device

Возвращает:

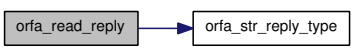
reply pointer or NULL if fails

См. определение в файле orfa.c строка

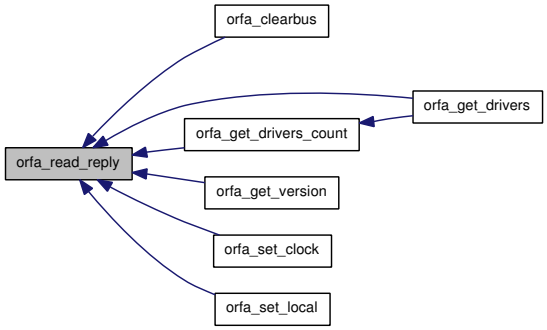
Перекрестные ссылки orfadev::fd, orfa_str_reply_type(), orfareply::size и orfadev::vars.

Используется в orfa_clearbus(), orfa_get_drivers(), orfa_get_drivers_count(), orfa_get_version(), orfa_set_clock() и orfa_set_local().

Граф вызовов:



Граф вызова функции:



6.3.1.2 const char* orfa_str_reply_type (struct orfareply * reply)

Stringify orfareply type

Аргументы:

← *reply Reply

Возвращает:

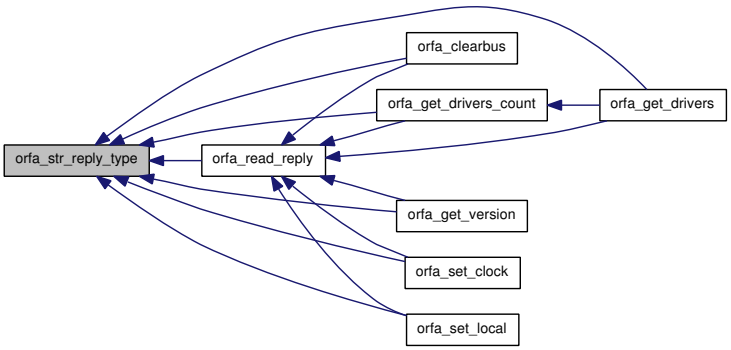
const string

См. определение в файле orfa.c строка

Перекрестные ссылки orfareply::OR_CLEAR, orfareply::OR_CLOCK, orfareply::OR_ERROR, orfareply::OR_LOCAL, orfareply::OR_READ, orfareply::OR_VERSION, orfareply::OR_WRITE и orfareply::type.

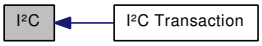
Используется в orfa_clearbus(), orfa_get_drivers(), orfa_get_drivers_count(), orfa_get_version(), orfa_read_reply(), orfa_set_clock() и orfa_set_local().

Граф вызова функции:



6.4 I²C

Граф связей класса I²C:



Группы

- I²C Transaction

Функции

- bool orfa_write (struct orfadev *device, int16_t address, uint8_t *data, uint8_t size)
- bool orfa_read (struct orfadev *device, int16_t address, uint8_t size)

6.4.1 Функции

6.4.1.1 bool orfa_read (struct orfadev * device, int16_t address, uint8_t size)

Read request

Аргументы:

← *device pointer to device

← address if < 0 then used local address

← read bytes count

Возвращает:

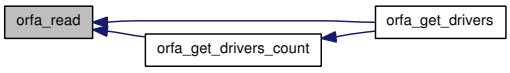
true if success, false if fails

См. определение в файле orfa.c строка

Перекрестные ссылки orfadev::fd, orfadev::in_transaction и orfadev::local.

Используется в orfa_get_drivers() и orfa_get_drivers_count().

Граф вызова функции:



6.4.1.2 bool orfa_write (struct orfadev * device, int16_t address, uint8_t * data, uint8_t size)

Write request

Аргументы:

← ∗device pointer to device

← address if < 0 then used local address

← ∗data pointer to data block

← size size of data block

Возвращает:

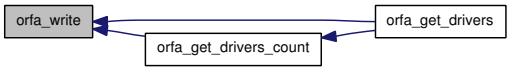
true if success, false if fails

См. определение в файле orfa.c строка

Перекрестные ссылки orfadev::fd, orfadev::in_transaction и orfadev::local.

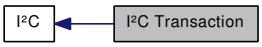
Используется в orfa_get_drivers() и orfa_get_drivers_count().

Граф вызова функции:



6.5 I²C Transaction

Граф связей класса I²C Transaction:



Функции

- void orfa_tr_start (struct orfadev *device)
- bool orfa_tr_commit (struct orfadev *device)

6.5.1 Подробное описание

I²C transaction is a command ‘S ... [S ...[S ...]] P‘

6.5.2 Функции

6.5.2.1 bool orfa_tr_commit (struct orfadev * device)

Commit transaction

Аргументы:

← *device pointer to device

Возвращает:

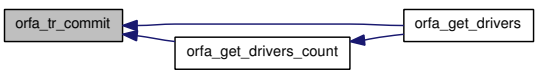
true if success, false if fails

См. определение в файле orfa.c строка

Перекрестные ссылки orfadev::fd и orfadev::in_transaction.

Используется в orfa_get_drivers() и orfa_get_drivers_count().

Граф вызова функции:



6.5.2.2 void orfa_tr_start (struct orfadev * device)

Start I²C transaction

Аргументы:

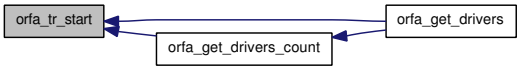
← *device pointer to device

См. определение в файле orfa.c строка

Перекрестные ссылки orfadev::in_transaction.

Используется в orfa_get_drivers() и orfa_get_drivers_count().

Граф вызова функции:



6.6 Special Requests

Функции

- char * orfa_get_version (struct orfadev *device)
- int orfa_get_clock (struct orfadev *device)
- bool orfa_set_clock (struct orfadev *device, uint16_t clock)
- uint8_t orfa_get_local (struct orfadev *device)
- bool orfa_set_local (struct orfadev *device, uint8_t local)
- bool orfa_clearbus (struct orfadev *device)

6.6.1 Функции

6.6.1.1 bool orfa_clearbus (struct orfadev * device)

Clear I²C bus

Аргументы:

← *device pointer to device

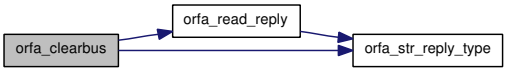
Возвращает:

true if success, false if fails

См. определение в файле orfa.c строка

Перекрестные ссылки orfadev::fd, orfareply::OR_CLEAR, orfa_read_reply(), orfa_str_reply_type() и orfareply::type.

Граф вызовов:



6.6.1.2 int orfa_get_clock (struct orfadev * device)

Get I²C clock

Аргументы:

← *device pointer to device

Возвращает:

current clock

См. определение в файле orfa.c строка

Перекрестные ссылки orfadev::clock.

6.6.1.3 uint8_t orfa_get_local (struct orfadev * device)

Get ORFA local I²C address

Аргументы:

← *device pointer to device

Возвращает:

current local address

См. определение в файле orfa.c строка

Перекрестные ссылки orfadev::local.

6.6.1.4 char* orfa_get_version (struct orfadev * device)

Get protocol version Don't foget free() result.

Аргументы:

← *device pointer to device

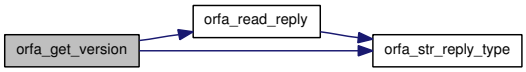
Возвращает:

string ptr if success, NULL if fails

См. определение в файле orfa.c строка

Перекрестные ссылки orfareply::data, orfadev::fd, orfareply::OR_VERSION, orfa_read_reply(), orfa_str_reply_type(), orfareply::size и orfareply::type.

Граф вызовов:



6.6.1.5 bool orfa_set_clock (struct orfadev * device, uint16_t clock)

Set I²C clock

Аргументы:

← *device pointer to device

← clock new clock in kHz

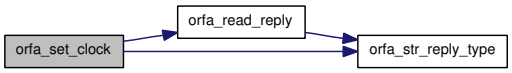
Возвращает:

true if success, false if fails

См. определение в файле orfa.c строка

Перекрестные ссылки orfadev::clock, orfareply::data, orfadev::fd, orfareply::OR_CLOCK, orfa_read_reply(), orfa_str_reply_type(), orfareply::size и orfareply::type.

Граф вызовов:



6.6.1.6 bool orfa_set_local (struct orfadev * device, uint8_t local)

Set ORFA local I²C address

Аргументы:

← *device pointer to device

local new address

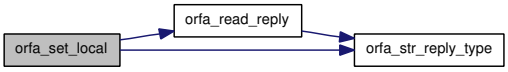
Возвращает:

true if success, false if fails

См. определение в файле orfa.c строка

Перекрестные ссылки orfareply::data, orfadev::fd, orfadev::local, orfareply::OR_LOCAL, orfa_read_reply(), orfa_str_reply_type(), orfareply::size и orfareply::type.

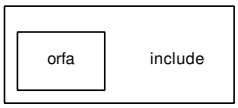
Граф вызовов:



Глава 7

Директории

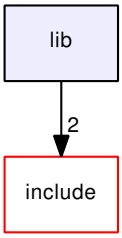
7.1 Содержание директории include/



Директории

- директория orfa

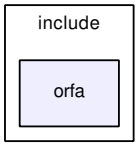
7.2 Содержание директории lib/



Файлы

- файл intro.c
- файл orfa.c
- файл serial.h
- файл serial_posix.c
- файл serial_win.c

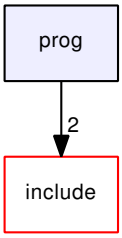
7.3 Содержание директории include/orfa/



Файлы

- файл intro.h
- файл orfa.h

7.4 Содержание директории prog/



Файлы

- файл lsorfa.c

Глава 8

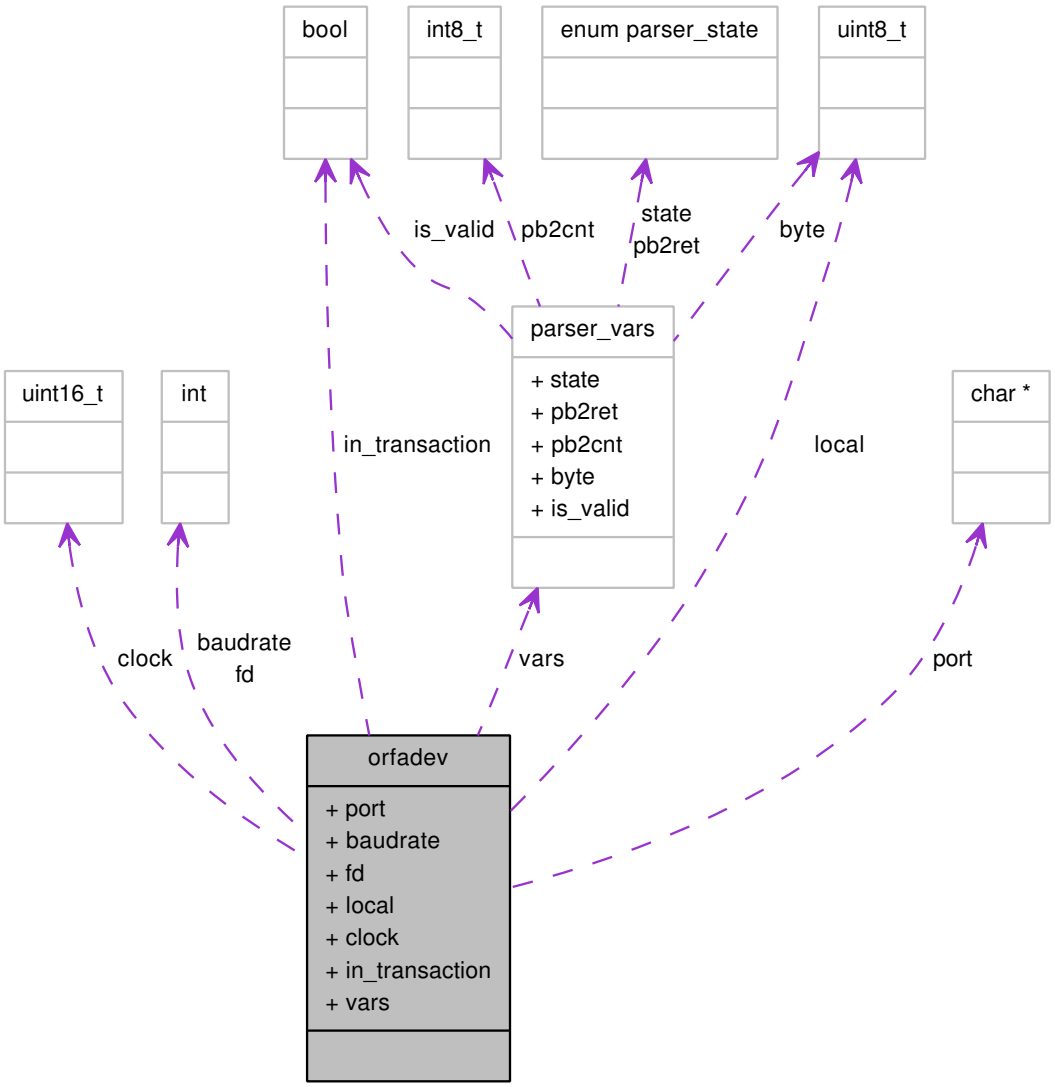
Структуры данных

8.1 Структура orfadev

Device structure.

#include <orfa.h>

Граф связей класса orfadev:



Поля данных

- char * port

  Serial device string.

- int baudrate

  Serial baud rate (115200 etc.).

- FD_TYPE fd

  Opened file descriptor.

- uint8_t local

  Local I²C device address.

- uint16_t clock
  I²C clock speed (kHz).
- bool in_transaction
  Transaction started.
- struct parser_vars vars
  Parser internal vars.

8.1.1 Подробное описание

Device structure.

См. определение в файле orfa.h строка

Объявления и описания членов структуры находятся в файле:

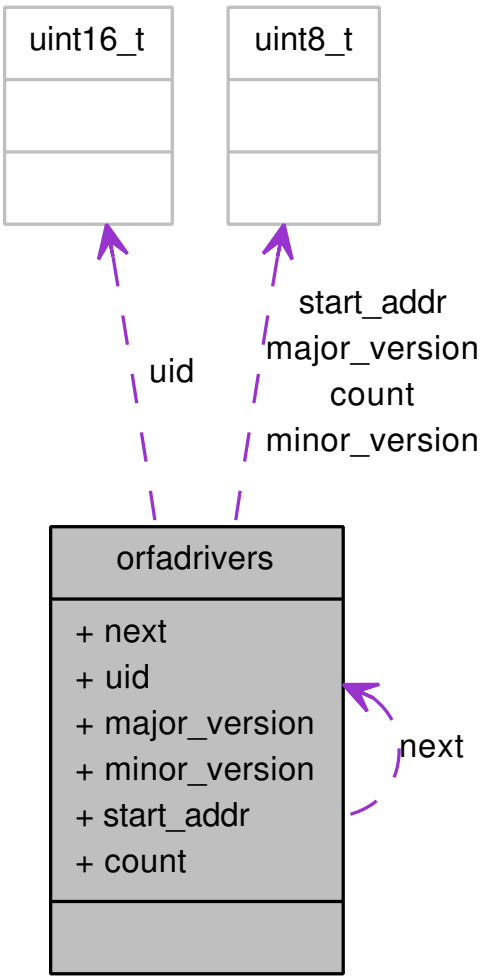
- include/orfa/orfa.h

8.2 Структура orfadrivers

Driver list.

#include <intro.h>

Граф связей класса orfadrivers:



Поля данных

- uint16_t uid
  Driver Unique ID (UID).
- uint8_t major_version
  Major Version.
- uint8_t minor_version
  Minor Version.
- uint8_t start_addr

First register address.

- uint8_t count
  Registers count.

8.2.1 Подробное описание

Driver list.

См. определение в файле intro.h строка

Объявления и описания членов структуры находятся в файле:

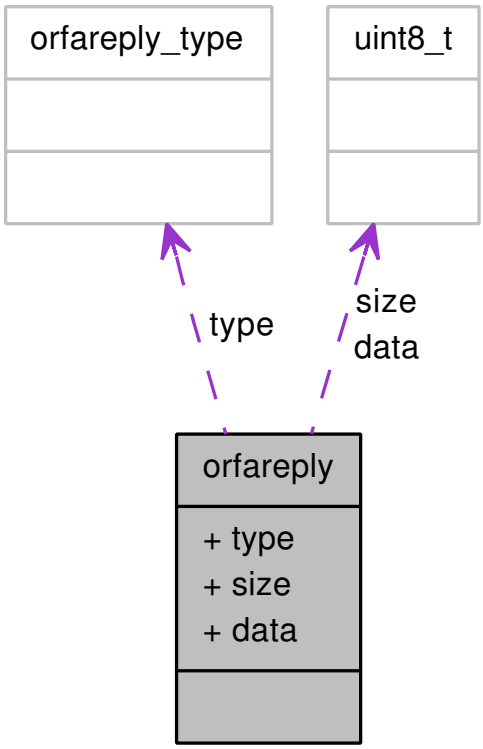
- include/orfa/intro.h

8.3 Структура orfareply

Reply structure.

#include <orfa.h>

Граф связей класса orfareply:



Открытые типы

- enum orfareply_type {
  OR_WRITE, OR_READ, OR_VERSION, OR_CLOCK,
  OR_LOCAL, OR_CLEAR, OR_ERROR }
  Reply type.

Поля данных

- enum orfareply::orfareply_type type
  Reply type.
- uint8_t size
  Reply size (0 < size < REPLY_DATA_SIZE).
- uint8_t data [REPLY_DATA_SIZE]
  Reply data.

8.3.1 Подробное описание

Reply structure.

См. определение в файле orfa.h строка

8.3.2 Перечисления

8.3.2.1 enum orfareply_type

Reply type.

Элементы перечислений:

OR_WRITE I²C write reply (SW command).
OR_READ I²C read reply (SR command).
OR_VERSION Get Version reply (V command).
OR_CLOCK Set I²C Clock reply (C command).
OR_LOCAL Set ORFA's core slave I²C address (L command).
OR_CLEAR Clear Bus reply (X command).
OR_ERROR ERROR reply (if error expected).

См. определение в файле orfa.h строка

Объявления и описания членов структуры находятся в файле:

- include/orfa/orfa.h

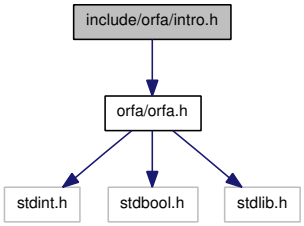
Глава 9

Файлы

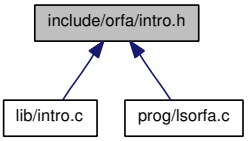
9.1 Файл include/orfa/intro.h

#include <orfa/orfa.h>

Граф включаемых заголовочных файлов для intro.h:



Граф файлов, в которые включается этот файл:



Структуры данных

- struct orfadrivers

  Driver list.

Функции

- uint8_t orfa_get_drivers_count (struct orfadev *device, int16_t address)
- struct orfadrivers * orfa_get_drivers (struct orfadev *device, int16_t address)
- bool orfa_free_drivers (struct orfadrivers **drivers)

9.1.1 Подробное описание

ORFA Introspection driver

Автор:

Vladimir Ermakov

См. определение в файле intro.h

9.2 intro.h

```
00001 /*
00002  *  libORFA -- PC-side gate libruary for Open Robotics Firmware Architecture
00003  *
00004  *  Copyright (c) 2009 Vladimir Ermakov
00005  *
00006  *  Permission is hereby granted, free of charge, to any person obtaining a copy
00007  *  of this software and associated documentation files (the "Software"), to deal

00008  *  in the Software without restriction, including without limitation the rights
00009  *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  *  copies of the Software, and to permit persons to whom the Software is
00011  *  furnished to do so, subject to the following conditions:
00012  *
00013  *  The above copyright notice and this permission notice shall be included in
00014  *  all copies or substantial portions of the Software.
00015  *
00016  *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017  *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018  *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019  *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020  *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

00021  *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00022  *  THE SOFTWARE.
00023  ******************************************************************************/
00029 #ifndef LIBORFAINTRO_H
00030 #define LIBORFAINTRO_H
00031
00032 #include <orfa/orfa.h>
00033
00039
00040 struct orfadrivers {
00041     // private
00042     struct orfadrivers *next;
00043     // public
00044     uint16_t uid;
00045     uint8_t major_version;
00046     uint8_t minor_version;
00047     uint8_t start_addr;
00048     uint8_t count;
00049 };
00050
00058 uint8_t orfa_get_drivers_count(struct orfadev *device, int16_t address);
00059
00067 struct orfadrivers *orfa_get_drivers(struct orfadev *device, int16_t address);
00068
00073 bool orfa_free_drivers(struct orfadrivers **drivers);
00074
00076
00077 #endif // LIBORFAINTRO_H
00078
```

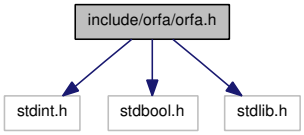
9.3 Файл include/orfa/orfa.h

#include <stdint.h>

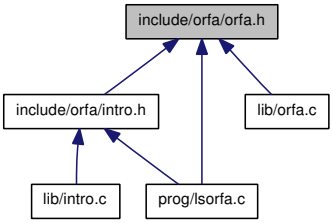
#include <stdbool.h>

#include <stdlib.h>

Граф включаемых заголовочных файлов для orfa.h:



Граф файлов, в которые включается этот файл:



Структуры данных

- struct parser_vars
- struct orfadev

  Device structure.

- struct orfareply

  Reply structure.

Макросы

- #define REPLY_DATA_SIZE 65

  Maximum reply size (ORFA's buffers CBF_SIZE=128, BUF_LEN=65).

Перечисления

- enum parser_state {
  PCOMMAND, PSTART, PSWACK, PBYTE1,
  PBYTE1_OR_STOP, PBYTE2, PVERSION, PERROR,
  PEOL }

Функции

- bool orfa_write (struct orfadev *device, int16_t address, uint8_t *data, uint8_t size)
- bool orfa_read (struct orfadev *device, int16_t address, uint8_t size)
- void orfa_tr_start (struct orfadev *device)
- bool orfa_tr_commit (struct orfadev *device)
- char * orfa_get_version (struct orfadev *device)
- int orfa_get_clock (struct orfadev *device)
- bool orfa_set_clock (struct orfadev *device, uint16_t clock)
- uint8_t orfa_get_local (struct orfadev *device)
- bool orfa_set_local (struct orfadev *device, uint8_t local)
- bool orfa_clearbus (struct orfadev *device)

- struct orfadev * orfa_open (char *port, int baudrate)
- int orfa_close (struct orfadev **device)

- struct orfareply * orfa_read_reply (struct orfadev *device)
- const char * orfa_str_reply_type (struct orfareply *reply)

9.3.1 Подробное описание

ORFA

Автор:

Vladimir Ermakov

См. определение в файле orfa.h

9.3.2 Перечисления

9.3.2.1 enum parser_state

Элементы перечислений:

PCOMMAND Detect command.

PSTART Parse S command reply.

PSWACK SW Ack counter.

PBYTE1 First nibble.

PBYTE1_OR_STOP First nibble or stop.

PBYTE2 Second nibble.

PVERSION Parse V command reply.

PERROR Parse ERROR reply.

PEOL Wait end of line.

См. определение в файле orfa.h строка

9.4 orfa.h

```
00001 /*
00002  *  libORFA -- PC-side gate libruary for Open Robotics Firmware Architecture
00003  *
00004  *  Copyright (c) 2009 Vladimir Ermakov
00005  *
00006  *  Permission is hereby granted, free of charge, to any person obtaining a copy
00007  *  of this software and associated documentation files (the "Software"), to deal

00008  *  in the Software without restriction, including without limitation the rights
00009  *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  *  copies of the Software, and to permit persons to whom the Software is
00011  *  furnished to do so, subject to the following conditions:
00012  *
00013  *  The above copyright notice and this permission notice shall be included in
00014  *  all copies or substantial portions of the Software.
00015  *
00016  *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017  *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019  *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

00021  *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00022  *  THE SOFTWARE.
00023  ******************************************************************************/
00029 #ifndef LIBORFA_H
00030 #define LIBORFA_H
00031
00032 #include <stdint.h>
00033 #include <stdbool.h>
00034 #include <stdlib.h>
00035
00036 #ifdef WIN32
00037 # include <windows.h>
00038 typedef HANDLE FD_TYPE;
00039 # define INVALID_FD_VAL INVALID_HANDLE_VALUE
00040 #else
00041 typedef int FD_TYPE;
00042 # define INVALID_FD_VAL -1
00043 #endif
00044
00045 #ifdef __cplusplus
00046 extern "C" {
00047 #endif
00048
00049 // -- Private types --
00050
00051 enum parser_state {
00052     PCOMMAND,
00053     PSTART,
00054     PSWACK,
00055     PBYTE1,
00056     PBYTE1_OR_STOP,
00057     PBYTE2,
00058     PVERSION,
00059     PERROR,
00060     PEOL
00061 };
00062
00063 struct parser_vars {
00064     enum parser_state state;
00065     enum parser_state pb2ret;
00066     int8_t pb2cnt;
00067     uint8_t byte;
00069     bool is_valid;
```

```
00070 };
00071
00072 // -- Public types --
00073
00079
00080 struct orfadev {
00081     // public
00082     char *port;
00083     int baudrate;
00084     // private
00085     FD_TYPE fd;
00086     uint8_t local;
00087     uint16_t clock;
00088     bool in_transaction;
00089     struct parser_vars vars;
00090 };
00091
00093
00099
00100 #define REPLY_DATA_SIZE 65
00101
00103 struct orfareply {
00105     enum orfareply_type {
00106         OR_WRITE,
00107         OR_READ,
00108         OR_VERSION,
00109         OR_CLOCK,
00110         OR_LOCAL,
00111         OR_CLEAR,
00112         OR_ERROR,
00113     } type;
00114     uint8_t size;
00115     uint8_t data[REPLY_DATA_SIZE];
00116 };
00117
00119
00120 // -- Public functions --
00121
00131 struct orfadev *orfa_open(char *port, int baudrate);
00132
00137 int orfa_close(struct orfadev **device);
00138
00140
00153 bool orfa_write(struct orfadev *device, int16_t address, uint8_t *data,
00154         uint8_t size);
00155
00162 bool orfa_read(struct orfadev *device, int16_t address, uint8_t size);
00163
00172 void orfa_tr_start(struct orfadev *device);
00173
00178 bool orfa_tr_commit(struct orfadev *device);
00179
00182
00192 struct orfareply *orfa_read_reply(struct orfadev *device);
00193
00198 const char *orfa_str_reply_type(struct orfareply *reply);
00199
00201
00212 char *orfa_get_version(struct orfadev *device);
00213
00218 int orfa_get_clock(struct orfadev *device);
00219
00225 bool orfa_set_clock(struct orfadev *device, uint16_t clock);
00226
00231 uint8_t orfa_get_local(struct orfadev *device);
00232
00238 bool orfa_set_local(struct orfadev *device, uint8_t local);
```

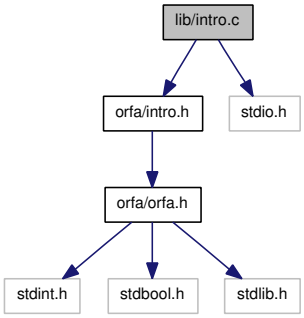
```
00239
00244 bool orfa_clearbus(struct orfadev *device);
00245
00247
00248 #ifdef __cplusplus
00249 }; // extern "C"
00250 #endif
00251
00252 #endif // LIBORFA_H
00253
```

9.5 Файл lib/intro.c

#include <orfa/intro.h>

#include <stdio.h>

Граф включаемых заголовочных файлов для intro.c:



Функции

- uint8_t orfa_get_drivers_count (struct orfadev *device, int16_t address)
- struct orfadrivers * orfa_get_drivers (struct orfadev *device, int16_t address)
- bool orfa_free_drivers (struct orfadrivers **drivers)

9.5.1 Подробное описание

ORFA Introspection driver See details here: http://roboforum.ru/wiki/ORFA_Introspection_-driver

Автор:

Vladimir Ermakov

См. определение в файле intro.c

9.6 intro.c

```
00001 /*
00002  *  libORFA -- PC-side gate libruary for Open Robotics Firmware Architecture
00003  *
00004  *  Copyright (c) 2009 Vladimir Ermakov
00005  *
00006  *  Permission is hereby granted, free of charge, to any person obtaining a copy
00007  *  of this software and associated documentation files (the "Software"), to deal

00008  *  in the Software without restriction, including without limitation the rights
00009  *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  *  copies of the Software, and to permit persons to whom the Software is
00011  *  furnished to do so, subject to the following conditions:
00012  *
00013  *  The above copyright notice and this permission notice shall be included in
00014  *  all copies or substantial portions of the Software.
00015  *
00016  *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017  *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019  *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

00021  *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00022  *  THE SOFTWARE.
00023  ******************************************************************************/
00031 #include <orfa/intro.h>
00032 #include <stdio.h>
00033
00034 // introspection register always zero
00035 #define INTRO 0x00
00036
00037 uint8_t orfa_get_drivers_count(struct orfadev *device, int16_t address)
00038 {
00039     struct orfareply *reply;
00040     uint8_t buf[] = {INTRO, 0};
00041     uint8_t count;
00042
00043     if (device == NULL) {
00044         perror("pointer error!\n");
00045         return 0;
00046     }
00047
00048     // transaction
00049     orfa_tr_start(device);
00050     if (!orfa_write(device, address, buf, 2)) {
00051         perror("get_drivers_count: write return false!\n");
00052         return 0;
00053     }
00054     if (!orfa_read(device, address, 1)) {
00055         perror("get_drivers_count: read return false!\n");
00056         return 0;
00057     }
00058     if (!orfa_tr_commit(device)) {
00059         perror("get_drivers_count: commit return false!\n");
00060         return 0;
00061     }
00062     // end transaction
00063
00064     if ((reply = orfa_read_reply(device)) == NULL) {
00065         perror("get_drivers_count: no reply\n");
00066         return 0;
00067     }
00068     if (reply->type != OR_WRITE) {
00069         fprintf(stderr, "get_drivers_count: incorrect reply type: %s\n",
00070                 orfa_str_reply_type(reply));
```

```
00071         free(reply);
00072         return 0;
00073     }
00074     free(reply);
00075
00076     if ((reply = orfa_read_reply(device)) == NULL) {
00077         perror("get_drivers_count: no reply\n");
00078         return 0;
00079     }
00080     if (reply->type != OR_READ) {
00081         fprintf(stderr, "get_drivers_count: incorrect reply type: %s\n",
00082                 orfa_str_reply_type(reply));
00083         free(reply);
00084         return 0;
00085     }
00086     if (reply->size != 1) {
00087         fprintf(stderr, "get_drivers_count: wrong reply size: %i\n",
00088                 reply->size);
00089         free(reply);
00090         return 0;
00091     }
00092
00093     count = reply->data[0];
00094     free(reply);
00095
00096     return count;
00097 }
00098
00099 /* Calculated for 128 char / 64 byte ORFA's cbf_t
00100  * - size of one pkg: 6 byte (12 chars)
00101  * - reply header and end (SWAASR ... P\n): 8 chars
00102  */
00103 #define MAX_PER_PKG 10
00104 #define PKG_LEN 6
00105
00106 struct orfadrivers *orfa_get_drivers(struct orfadev *device, int16_t address)
00107 {
00108     struct orfadrivers *head=NULL, *driver;
00109     struct orfareply *reply;
00110     uint8_t count;
00111     uint8_t buf[] = {INTRO, 1};
00112
00113     if ((count=orfa_get_drivers_count(device, address)) == 0) {
00114         perror("get_drivers: count == 0\n");
00115         return NULL;
00116     }
00117
00118     uint8_t max_cnt = count / MAX_PER_PKG;
00119     uint8_t min_cnt = count % MAX_PER_PKG;
00120     uint8_t min_num = MAX_PER_PKG * max_cnt + 1;
00121     uint8_t reply_cnt = max_cnt*2 + (min_cnt)? 2:0;
00122
00123     // read maximum long packets
00124     for (int i=0; i<max_cnt; i++) {
00125         buf[1] = MAX_PER_PKG * i + 1;
00126         // transaction
00127         orfa_tr_start(device);
00128         if (!orfa_write(device, address, buf, 2)) {
00129             return NULL;
00130         }
00131         if (!orfa_read(device, address, MAX_PER_PKG * PKG_LEN)) {
00132             return NULL;
00133         }
00134         if (!orfa_tr_commit(device)) {
00135             return NULL;
00136         }
00137         // end transaction
```

```
00138     }
00139
00140     // read last elements
00141     buf[1] = min_num;
00142     // transaction
00143     orfa_tr_start(device);
00144     if (!orfa_write(device, address, buf, 2)) {
00145         return NULL;
00146     }
00147     if (!orfa_read(device, address, min_cnt * PKG_LEN)) {
00148         return NULL;
00149     }
00150     if (!orfa_tr_commit(device)) {
00151         return NULL;
00152     }
00153     // end transaction
00154
00155     for (int i=0; i < reply_cnt; i++) {
00156         if ((reply = orfa_read_reply(device)) == NULL) {
00157             perror("get_drivers: read_reply() return NULL!\n");
00158             return head;
00159         }
00160
00161         if (reply->type == OR_READ) {
00162             if (reply->size % 6) {
00163                 fprintf(stderr, "get_drivers: wrong read reply size: %i\n",
00164                         reply->size);
00165             } else {
00166                 for (int pi=0; pi < reply->size; pi += 6) {
00167                     if ((driver = malloc(sizeof(struct orfadrivers)))
00168                             == NULL) {
00169                         perror("Out of memory!\n");
00170                         return head;
00171                     }
00172
00173                     driver->next = head;
00174                     driver->uid =           (reply->data[pi+0] << 8)
00175                                 |           (reply->data[pi+1]);
00176                     driver->major_version = reply->data[pi+2];
00177                     driver->minor_version = reply->data[pi+3];
00178                     driver->start_addr =    reply->data[pi+4];
00179                     driver->count =         reply->data[pi+5];
00180                     head = driver;
00181                 }
00182             }
00183         } else if (reply->type != OR_WRITE) {
00184             fprintf(stderr, "get_drivers: wrong reply type: %s\n",
00185                     orfa_str_reply_type(reply));
00186         }
00187
00188         free(reply);
00189     }
00190
00191     return head;
00192 }
00193
00194 bool orfa_free_drivers(struct orfadrivers **drivers)
00195 {
00196     struct orfadrivers *head;
00197
00198     if (drivers == NULL) {
00199         perror("pointer error!\n");
00200         return false;
00201     }
00202     if (*drivers == NULL) {
00203         perror("pointer error!\n");
00204         return false;
```

```
00205     }
00206
00207     do {
00208         head = (*drivers)->next;
00209         free(*drivers);
00210         *drivers = head;
00211     } while (head != NULL);
00212
00213     *drivers = NULL;
00214
00215     return true;
00216 }
00217
```

9.7 Файл lib/orfa.c

#include <orfa/orfa.h>

#include "serial.h"

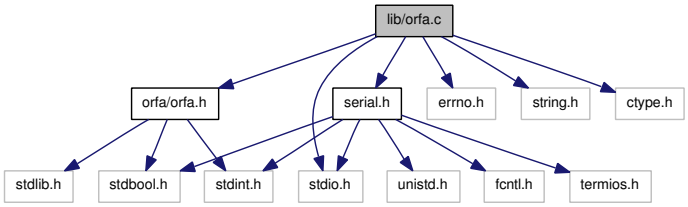
#include <stdio.h>

#include <errno.h>

#include <string.h>

#include <ctype.h>

Граф включаемых заголовочных файлов для orfa.c:



Функции

- struct orfadev * orfa_open (char *port, int baudrate)
- int orfa_close (struct orfadev **device)
- void orfa_tr_start (struct orfadev *device)
- bool orfa_tr_commit (struct orfadev *device)
- bool orfa_write (struct orfadev *device, int16_t address, uint8_t *data, uint8_t size)
- bool orfa_read (struct orfadev *device, int16_t address, uint8_t size)
- struct orfareply * orfa_read_reply (struct orfadev *device)
- const char * orfa_str_reply_type (struct orfareply *reply)
- char * orfa_get_version (struct orfadev *device)
- int orfa_get_clock (struct orfadev *device)
- bool orfa_set_clock (struct orfadev *device, uint16_t clock)
- uint8_t orfa_get_local (struct orfadev *device)
- bool orfa_set_local (struct orfadev *device, uint8_t local)
- bool orfa_clearbus (struct orfadev *device)

9.7.1 Подробное описание

ORFA

Автор:

Vladimir Ermakov

Необходимо сделать

Test timeouts

См. определение в файле orfa.c

9.7.2 Функции

9.7.2.1 int orfa_close (struct orfadev ** device)

Close device and free memory.

Аргументы:

← **device pointer to device pointer

Возвращает:

close() ret

См. определение в файле orfa.c строка

9.7.2.2 const char* orfa_str_reply_type (struct orfareply * reply)

Stringify orfareply type

Аргументы:

← *reply Reply

Возвращает:

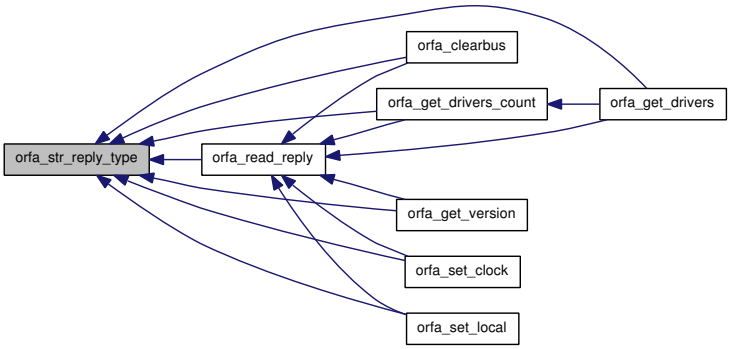
const string

См. определение в файле orfa.c строка

Перекрестные ссылки orfareply::OR_CLEAR, orfareply::OR_CLOCK, orfareply::OR_ERROR, orfareply::OR_LOCAL, orfareply::OR_READ, orfareply::OR_VERSION, orfareply::OR_WRITE и orfareply::type.

Используется в orfa_clearbus(), orfa_get_drivers(), orfa_get_drivers_count(), orfa_get_version(), orfa_read_reply(), orfa_set_clock() и orfa_set_local().

Граф вызова функции:



9.8 orfa.c

```
00001 /*
00002  *  libORFA -- PC-side gate libruary for Open Robotics Firmware Architecture
00003  *
00004  *  Copyright (c) 2009 Vladimir Ermakov
00005  *
00006  *  Permission is hereby granted, free of charge, to any person obtaining a copy
00007  *  of this software and associated documentation files (the "Software"), to deal

00008  *  in the Software without restriction, including without limitation the rights
00009  *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  *  copies of the Software, and to permit persons to whom the Software is
00011  *  furnished to do so, subject to the following conditions:
00012  *
00013  *  The above copyright notice and this permission notice shall be included in
00014  *  all copies or substantial portions of the Software.
00015  *
00016  * THE SOFTWARE IS PEOLOVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPEOLESS O
      R
00017  *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018  * FITNESS FOR A PSWACKRTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL T
      HE
00019  *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

00021  *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00022  *  THE SOFTWARE.
00023  ******************************************************************************/
00031 #include <orfa/orfa.h>
00032 #include "serial.h"
00033
00034 #include <stdio.h>
00035 #include <errno.h>
00036 #include <string.h>
00037 #include <ctype.h>
00038
00039 #define CHECK_POINTER_R(p, r)       \
00040     if (p == NULL) {                \
00041         perror("pointer error!\n"); \
00042         return r;                   \
00043     }
00044
00045 #define CHECK_POINTER_RF(p) CHECK_POINTER_R(p, false)
00046 #define CHECK_POINTER_RN(p) CHECK_POINTER_R(p, NULL)
00047
00048 #define CHECK_ALLOC_R(p, r)                 \
00049     if (p == NULL) {                        \
00050         perror("Out of memory!\n");         \
00051         return r;                           \
00052     }
00053
00054 #define CHECK_ALLOC_RN(p) CHECK_ALLOC_R(p, NULL)
00055
00056 // == Private ==
00057 // -- private char convert functions --
00058
00063 static int8_t xtoi(uint8_t c)
00064 {
00065     if((c >= '0') && (c <= '9')) {
00066         return c - '0';
00067     } else if((c >= 'A')&&(c <= 'F')) {
00068         return c - 'A' + 10;
00069     } else {
00070         return -1;
00071     }
00072 }
```

```
00073
00078 static char itox(uint8_t c)
00079 {
00080     char ret;
00081
00082     if (c <= 0x09) {
00083         ret = '0' + c;
00084     } else if (c <= 0x0f) {
00085         ret = 'A' + c - 10;
00086     } else {
00087         ret = 'X';
00088     }
00089     return ret;
00090 }
00091
00092 // -- parser --
00093
00094 #define REPLY_DATA_ADD(_reply, _data) \
00095     (_reply)->data[(_reply)->size] = (_data); \
00096     if ((_reply)->size < REPLY_DATA_SIZE-1) { \
00097         (_reply)->size += 1; \
00098     }
00099
00106 static bool
00107 parse(struct parser_vars *self, uint8_t c, struct orfareply *reply)
00108 {
00109     int8_t tmp;
00110
00111     if (self == NULL || reply == NULL) {
00112         perror("pointer error!\n");
00113         return false;
00114     }
00115
00116     // clean input
00117     c = toupper(c);
00118     c = (c == '\r')? '\n' : c;
00119
00120     switch (self->state) {
00121         case PCOMMAND:
00122             self->pb2cnt = 1;
00123             self->is_valid = true;
00124             reply->size = 0;
00125
00126             switch (c) {
00127                 case 'S':
00128                     self->state = PSTART;
00129                     self->pb2cnt = -1;
00130                     break;
00131
00132                 case 'C':
00133                     self->state = PBYTE1;
00134                     self->pb2cnt = 2;
00135                     reply->type = OR_CLOCK;
00136                     break;
00137
00138                 case 'L':
00139                     self->state = PBYTE1;
00140                     reply->type = OR_LOCAL;
00141                     break;
00142
00143                 case 'X':
00144                     self->state = PEOL;
00145                     reply->type = OR_CLEAR;
00146                     reply->size = 1;
00147                     reply->data[0] = true;
00148                     break;
00149
```

```
00150             case 'V':
00151                 self->state = PVERSION;
00152                 reply->type = OR_VERSION;
00153                 break;
00154 
00155             case 'E':
00156                 self->state = PERROR;
00157                 reply->type = OR_ERROR;
00158                 break;
00159 
00160             case '\n':
00161                 // skip blank line
00162                 break;
00163 
00164             default:
00165                 // unknown command
00166                 self->state = PEOL;
00167                 self->is_valid = false;
00168                 break;
00169         }
00170         break;
00171 
00172     case PSTART:
00173         switch (c) {
00174             case 'R':
00175                 self->state = PBYTE1_OR_STOP;
00176                 reply->type = OR_READ;
00177                 reply->size = 0;
00178                 break;
00179 
00180             case 'W':
00181                 self->state = PSWACK;
00182                 reply->type = OR_WRITE;
00183                 reply->size = 0;
00184                 break;
00185 
00186             case '\n':
00187                 // P expected
00188                 self->is_valid = false;
00189                 self->state = PCOMMAND;
00190                 break;
00191 
00192             default:
00193                 self->is_valid = false;
00194                 self->state = PEOL;
00195                 break;
00196         }
00197         break;
00198 
00199     case PSWACK:
00200         switch (c) {
00201             case 'A':
00202                 REPLY_DATA_ADD(reply, true);
00203                 break;
00204 
00205             case 'P':
00206                 self->state = PEOL;
00207                 break;
00208 
00209             case 'S':
00210                 self->state = PSTART;
00211                 return true;
00212                 break;
00213 
00214             case '\n':
00215                 // P expected
00216                 self->is_valid = false;
```

```
00217                 self->state = PCOMMAND;
00218                 break;
00219
00220             default:
00221                 self->is_valid = false;
00222                 self->state = PEOL;
00223                 break;
00224         }
00225         break;
00226
00227     case PBYTE1:
00228     case PBYTE1_OR_STOP:
00229         self->pb2ret = self->state;
00230         tmp = xtoi(c);
00231         if (tmp != -1) {
00232             self->byte = tmp << 4;
00233             self->state = PBYTE2;
00234         } else if (self->state == PBYTE1_OR_STOP && c == 'S') {
00235             // Next S reply start (transaction)
00236             self->state = PSTART;
00237             return true;
00238         } else if (self->state == PBYTE1_OR_STOP && c == 'P') {
00239             // S reply end
00240             self->state = PEOL;
00241         } else if (c == '\n') {
00242             // P expected
00243             self->is_valid = false;
00244             self->state = PCOMMAND;
00245         } else {
00246             // error
00247             self->state = PEOL;
00248             self->is_valid = false;
00249         }
00250         break;
00251
00252     case PBYTE2:
00253         tmp = xtoi(c);
00254         if (tmp != -1) {
00255             self->byte |= tmp;
00256             REPLY_DATA_ADD(reply, self->byte);
00257
00258             // all bytes readed?
00259             self->pb2cnt -= (self->pb2cnt > 0)? 0 : 1;
00260             self->state = (self->pb2cnt == 0)? PEOL : self->pb2ret;
00261         } else if (c == '\n') {
00262             // P expected
00263             self->is_valid = false;
00264             self->state = PCOMMAND;
00265         } else {
00266             self->is_valid = false;
00267             self->state = PEOL;
00268         }
00269         break;
00270
00271     case PVERSION:
00272         if (c == '\n') {
00273             self->state = PCOMMAND;
00274             // gurantee that version string end at \0
00275             reply->data[REPLY_DATA_SIZE-1] = '\0';
00276             return true;
00277         }
00278         REPLY_DATA_ADD(reply, c);
00279         break;
00280
00281     case PERROR:
00282         switch (c) {
00283             case 'R':
```

```
00284                 case 'O':
00285                     // eRROR
00286                     break;
00287 
00288                 case ' ':
00289                     // error code start
00290                     self->state = PBYTE1;
00291                     break;
00292 
00293                 case '\n':
00294                     // P expected
00295                     self->is_valid = false;
00296                     self->state = PCOMMAND;
00297                     break;
00298 
00299                 default:
00300                     self->is_valid = false;
00301                     self->state = PEOL;
00302                     break;
00303             }
00304             break;
00305 
00306         case PEOL:
00307             if (c == '\n') {
00308                 self->state = PCOMMAND;
00309 
00310                 if (self->is_valid) {
00311                     return true;
00312                 }
00313             }
00314             break;
00315     }
00316 
00317     return false;
00318 }
00319 
00320 // == Public ==
00321 
00322 // Open device
00323 struct orfadev *orfa_open(char *port, int baudrate)
00324 {
00325     FD_TYPE fd = open_and_configure(port, baudrate);
00326     if (fd == INVALID_FD_VAL) {
00327         return NULL;
00328     }
00329 
00330     struct orfadev *device = malloc(sizeof(struct orfadev));
00331     CHECK_ALLOC_RN(device);
00332 
00333     // set data
00334     device->fd = fd;
00335     device->port = port;
00336     device->baudrate = baudrate;
00337     device->clock = 100;  // power-on default 100 kHz
00338     device->local = 0xAA; // power-on default
00339     device->in_transaction = false;
00340     device->vars.state = PCOMMAND;
00341 
00342     return device;
00343 }
00344 
00345 // Close device
00346 int orfa_close(struct orfadev **device)
00347 {
00348     CHECK_POINTER_R(device, -1);
00349     CHECK_POINTER_R(*device, -1);
00350 
```

```
00351     int ret = posix_close((*device)->fd);
00352     free(*device);
00353     *device = NULL;
00354 
00355     return ret;
00356 }
00357 
00358 // I²C transaction start
00359 void orfa_tr_start(struct orfadev *device)
00360 {
00361     CHECK_POINTER_R(device, );
00362 
00363     device->in_transaction = true;
00364 }
00365 
00366 // I²C transaction end
00367 bool orfa_tr_commit(struct orfadev *device)
00368 {
00369     CHECK_POINTER_RF(device);
00370 
00371     if (posix_write(device->fd, "P\n", 2) < 2) {
00372         perror("posix_write() failed!\n");
00373         return false;
00374     }
00375 
00376     device->in_transaction = false;
00377 
00378     return true;
00379 }
00380 
00381 // I²C write request
00382 bool orfa_write(struct orfadev *device, int16_t address,
00383         uint8_t *data, uint8_t size)
00384 {
00385     uint8_t bs[3], address8;
00386 
00387     CHECK_POINTER_RF(device);
00388 
00389     if (address < 0) {
00390         address8 = device->local;
00391     } else {
00392         address8 = ((uint8_t) address);
00393     }
00394     address8 &= 0xfe;
00395 
00396     bs[0] = 'S';
00397     bs[1] = itox(address8 >> 4);
00398     bs[2] = itox(address8 & 0xf);
00399     if (posix_write(device->fd, bs, 3) < 3) {
00400         perror("posix_write() failed!\n");
00401         return false;
00402     }
00403 
00404     for (uint8_t i=0; i<size; i++) {
00405         uint8_t b = data[i];
00406         bs[0] = itox(b >> 4);
00407         bs[1] = itox(b & 0xf);
00408         if (posix_write(device->fd, bs, 2) < 2) {
00409             perror("posix_write() failed!\n");
00410             return false;
00411         }
00412     }
00413 
00414     if (!device->in_transaction) {
00415         if (posix_write(device->fd, "P\n", 2) < 2) {
00416             perror("posix_write() failed!\n");
00417             return false;
```

```
00418         }
00419     }
00420
00421     return true;
00422 }
00423
00424 // I²C read request
00425 bool orfa_read(struct orfadev *device, int16_t address, uint8_t size)
00426 {
00427     uint8_t bs[5], address8;
00428
00429     CHECK_POINTER_RF(device);
00430
00431     if (address < 0) {
00432         address8 = device->local;
00433     } else {
00434         address8 = ((uint8_t) address);
00435     }
00436     address8 |= 0x01;
00437
00438     bs[0] = 'S';
00439     bs[1] = itox(address8 >> 4);
00440     bs[2] = itox(address8 & 0xf);
00441     bs[3] = itox(size >> 4);
00442     bs[4] = itox(size & 0xf);
00443
00444     if (posix_write(device->fd, bs, 5) < 5) {
00445         perror("posix_write() failed!\n");
00446         return false;
00447     }
00448
00449     if (!device->in_transaction) {
00450         if (posix_write(device->fd, "P\n", 2) < 2) {
00451             perror("posix_write() failed!\n");
00452             return false;
00453         }
00454     }
00455
00456     return true;
00457 }
00458
00459 // get request reply
00460 struct orfareply *orfa_read_reply(struct orfadev *device)
00461 {
00462     CHECK_POINTER_RN(device);
00463
00464     struct orfareply *reply=malloc(sizeof(struct orfareply));
00465     CHECK_ALLOC_RN(reply);
00466
00467     uint8_t buf;
00468     do {
00469         // TODO test timeout
00470         int iret = posix_read(device->fd, &buf, 1);
00471         if (iret < 1) {
00472             fprintf(stderr, "read_reply: posix_read() returns error: %s\n",
00473                     strerror(errno));
00474             free(reply);
00475             return NULL;
00476         }
00477     } while (!parse(&(device->vars), buf, reply) && device->vars.is_valid);
00478
00479     if (!device->vars.is_valid) {
00480         fprintf(stderr, "reply not valid! [type=%s, size=%i]\n",
00481                 orfa_str_reply_type(reply), reply->size);
00482         free(reply);
00483         return NULL;
00484     }
```

```
00485
00486     return reply;
00487 }
00488
00489 const char *orfa_str_reply_type(struct orfareply *reply)
00490 {
00491     CHECK_POINTER_R(reply, "PTRERR");
00492
00493     switch (reply->type) {
00494         case OR_WRITE:   return "OR_WRITE";
00495         case OR_READ:    return "OR_READ";
00496         case OR_VERSION: return "OR_VERSION";
00497         case OR_CLOCK:   return "OR_CLOCK";
00498         case OR_LOCAL:   return "OR_LOCAL";
00499         case OR_CLEAR:   return "OR_CLEAR";
00500         case OR_ERROR:   return "OR_ERROR";
00501         default:         return "TYPEERR";
00502     }
00503 }
00504
00505 // -- ORFA special requests --
00506
00507 // get protocol version string
00508 char *orfa_get_version(struct orfadev *device)
00509 {
00510     CHECK_POINTER_RN(device);
00511
00512     if (posix_write(device->fd, "V\n", 2) < 2) {
00513         perror("version: posix_write() failed!\n");
00514         return NULL;
00515     }
00516
00517     struct orfareply *reply=orfa_read_reply(device);
00518     if (reply == NULL) {
00519         perror("version: no reply\n");
00520         return NULL;
00521     }
00522
00523     if (reply->type != OR_VERSION) {
00524         fprintf(stderr, "version: wrong reply type: %s\n",
00525                 orfa_str_reply_type(reply));
00526         free(reply);
00527         return NULL;
00528     }
00529
00530     if (reply->size == 0) {
00531         perror("version: wrong reply size == 0\n");
00532         return NULL;
00533     }
00534
00535     char *version_str=malloc(reply->size+1);
00536     CHECK_ALLOC_RN(version_str);
00537
00538     // copy version string
00539     memcpy(version_str, &(reply->data), reply->size);
00540     version_str[reply->size+1] = '\0';
00541
00542     free(reply);
00543
00544     return version_str;
00545 }
00546
00547 // get i2c clock
00548 int orfa_get_clock(struct orfadev *device)
00549 {
00550     CHECK_POINTER_R(device, 0);
00551
```

```
00552     return device->clock;
00553 }
00554
00555 // set i2c clock
00556 bool orfa_set_clock(struct orfadev *device, uint16_t clock)
00557 {
00558     uint8_t bs[6];
00559
00560     CHECK_POINTER_RF(device);
00561
00562     bs[0] = 'C';
00563     bs[1] = itox(clock >> 12);
00564     bs[2] = itox((clock >> 8) & 0xf);
00565     bs[3] = itox((clock >> 4) & 0xf);
00566     bs[4] = itox(clock & 0xf);
00567     bs[5] = '\n';
00568     if (posix_write(device->fd, bs, 6) < 6) {
00569         perror("clock: posix_write() failed!\n");
00570         return false;
00571     }
00572
00573     //device->clock = clock;
00574     struct orfareply *reply=orfa_read_reply(device);
00575     if (reply == NULL) {
00576         perror("clock: no reply\n");
00577         return false;
00578     }
00579
00580     if (reply->type != OR_CLOCK) {
00581         fprintf(stderr, "clock: wrong reply type: %s\n",
00582                 orfa_str_reply_type(reply));
00583         free(reply);
00584         return false;
00585     }
00586
00587     if (reply->size != 2) {
00588         fprintf(stderr, "clock: wrong size: %i\n", reply->size);
00589         free(reply);
00590         return false;
00591     }
00592
00593     device->clock = (reply->data[0]<<8)|(reply->data[1]);
00594     free(reply);
00595
00596     return true;
00597 }
00598
00599 // get local device address
00600 uint8_t orfa_get_local(struct orfadev *device)
00601 {
00602     CHECK_POINTER_R(device, 0);
00603
00604     return device->local;
00605 }
00606
00607 // set local device address
00608 bool orfa_set_local(struct orfadev *device, uint8_t local)
00609 {
00610     uint8_t bs[4];
00611
00612     CHECK_POINTER_RF(device);
00613
00614     bs[0] = 'L';
00615     bs[1] = itox(local >> 4);
00616     bs[2] = itox(local & 0xf);
00617     bs[3] = '\n';
00618     if (posix_write(device->fd, bs, 4) < 4) {
```

```
00619         perror("local: posix_write() failed!\n");
00620         return false;
00621     }
00622 
00623     //device->local = local;
00624     struct orfareply *reply=orfa_read_reply(device);
00625     if (reply == NULL) {
00626         perror("local: no reply\n");
00627         return false;
00628     }
00629 
00630     if (reply->type != OR_LOCAL) {
00631         fprintf(stderr, "local: wrong reply type: %s\n",
00632                 orfa_str_reply_type(reply));
00633         free(reply);
00634         return false;
00635     }
00636 
00637     if (reply->size != 1) {
00638         fprintf(stderr, "local: wrong size: %i\n", reply->size);
00639         free(reply);
00640         return false;
00641     }
00642 
00643     device->local = reply->data[0];
00644     free(reply);
00645 
00646     return true;
00647 }
00648 
00649 // clear i2c bus
00650 bool orfa_clearbus(struct orfadev *device)
00651 {
00652     CHECK_POINTER_RF(device);
00653 
00654     if (posix_write(device->fd, "X\n", 2) < 2) {
00655         perror("clearbus: posix_write() failed!\n");
00656         return false;
00657     }
00658 
00659     struct orfareply *reply=orfa_read_reply(device);
00660     if (reply == NULL) {
00661         perror("clearbus: no reply\n");
00662         return false;
00663     }
00664 
00665     if (reply->type != OR_CLEAR) {
00666         fprintf(stderr, "clearbus: wrong reply type: %s\n",
00667                 orfa_str_reply_type(reply));
00668         free(reply);
00669         return false;
00670     }
00671 
00672     free(reply);
00673 
00674     return true;
00675 }
00676 
```

9.9 Файл lib/serial.h

#include <stdint.h>

#include <stdbool.h>

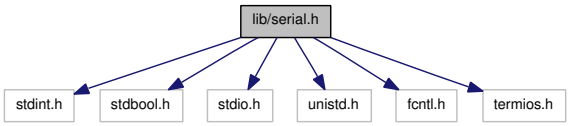
#include <stdio.h>

#include <unistd.h>

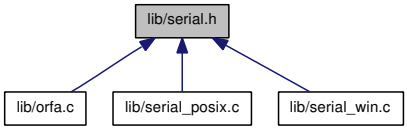
#include <fcntl.h>

#include <termios.h>

Граф включаемых заголовочных файлов для serial.h:



Граф файлов, в которые включается этот файл:



9.9.1 Подробное описание

Serial lib

Автор:

Vladimir Ermakov

См. определение в файле serial.h

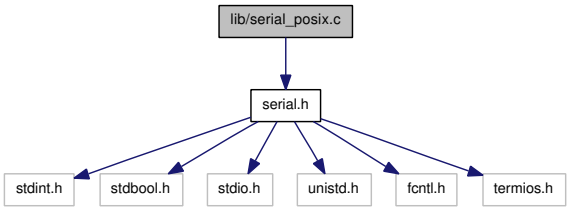
9.10 serial.h

```
00001 /*
00002  *  libORFA -- PC-side gate libruary for Open Robotics Firmware Architecture
00003  *
00004  *  Copyright (c) 2009 Vladimir Ermakov
00005  *
00006  *  Permission is hereby granted, free of charge, to any person obtaining a copy
00007  *  of this software and associated documentation files (the "Software"), to deal

00008  *  in the Software without restriction, including without limitation the rights
00009  *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  *  copies of the Software, and to permit persons to whom the Software is
00011  *  furnished to do so, subject to the following conditions:
00012  *
00013  *  The above copyright notice and this permission notice shall be included in
00014  *  all copies or substantial portions of the Software.
00015  *
00016 * THE SOFTWARE IS PEOLOVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPEOLESS O
      R
00017  *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 * FITNESS FOR A PSWACKRTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL T
      HE
00019  *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

00021  *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00022  *  THE SOFTWARE.
00023  ******************************************************************************/
00029 #ifndef SERIAL_H
00030 #define SERIAL_H
00031
00032 #include <stdint.h>
00033 #include <stdbool.h>
00034 #include <stdio.h>
00035
00036 #ifndef WIN32
00037 // == POSIX ==
00038 # include <unistd.h>
00039 # include <fcntl.h>
00040 # include <termios.h>
00041
00042 # define posix_close(fd) close(fd)
00043 # define posix_write(fd, buf, size) write(fd, buf, size)
00044 # define posix_read(fd, buf, size) read(fd, buf, size)
00045
00046 int open_and_configure(char *port, int baudrate);
00047
00048 #else
00049 // == WIN32 ==
00050 # include <windows.h>
00051
00052 # define posix_close(handle) CloseHandle(handle)
00053
00054 HANDLE open_and_configure(char *port, int baudrate);
00055 int posix_write(HANDLE handle, char *buf, size_t size);
00056 int posix_read(HANDLE handle, char *buf, size_t count);
00057
00058 #endif // !WIN32
00059
00060 #endif // SERIAL_H
00061
```

9.11 Файл lib/serial_posix.c

#include "serial.h"

Граф включаемых заголовочных файлов для serial_posix.c:



9.11.1 Подробное описание

Serial lib POSIX

Автор:

Vladimir Ermakov

См. определение в файле serial_posix.c

9.12 serial_posix.c

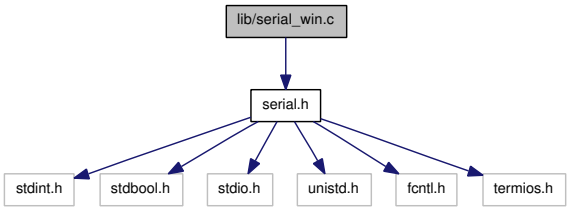
```
00001 /*
00002  *  libORFA -- PC-side gate libruary for Open Robotics Firmware Architecture
00003  *
00004  *  Copyright (c) 2009 Vladimir Ermakov
00005  *
00006  *  Permission is hereby granted, free of charge, to any person obtaining a copy
00007  *  of this software and associated documentation files (the "Software"), to deal

00008  *  in the Software without restriction, including without limitation the rights
00009  *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  *  copies of the Software, and to permit persons to whom the Software is
00011  *  furnished to do so, subject to the following conditions:
00012  *
00013  *  The above copyright notice and this permission notice shall be included in
00014  *  all copies or substantial portions of the Software.
00015  *
00016  * THE SOFTWARE IS PEOLOVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPEOLESS O
       R
00017  *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018  * FITNESS FOR A PSWACKRTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL T
       HE
00019  *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020  * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

00021  *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00022  *  THE SOFTWARE.
00023  ******************************************************************************/
00029 #include "serial.h"
00030
00031 static inline void setspeed(struct termios *options, int baudrate)
00032 {
00033     switch (baudrate) {
00034         case 9600:   baudrate = B9600;   break;
00035         case 115200: baudrate = B115200; break;
00036         default:
00037             fprintf(stderr, "Unsupported baudrate %i, set to 9600\n",
00038                     baudrate);
00039             baudrate = B9600;
00040             break;
00041     }
00042
00043     cfsetispeed(options, baudrate);
00044     cfsetospeed(options, baudrate);
00045 }
00046
00047 int open_and_configure(char *port, int baudrate)
00048 {
00049     struct termios options;
00050
00051     int fd = open(port, O_RDWR|O_NOCTTY|O_NDELAY);
00052     if (fd == -1) {
00053         fprintf(stderr, "open_and_configure: Unable to open %s\n", port);
00054         return -1;
00055     }
00056
00057     // set bloking
00058     fcntl(fd, F_SETFL, 0);
00059
00060     // Get the current options for the port...
00061     tcgetattr(fd, &options);
00062
00063     // Set the baud rate
00064     setspeed(&options, baudrate);
00065
00066     // Enable the receiver and set local mode, set 8N1
```

```
00067     options.c_cflag |= CLOCAL | CREAD;
00068     options.c_cflag &= ~PARENB;
00069     options.c_cflag &= ~CSTOPB;
00070     options.c_cflag &= ~CSIZE;
00071     options.c_cflag |= CS8;
00072 # ifdef CNEW_RTSCTS
00073     options.c_cflag &= ~CNEW_RTSCTS;
00074 # elif defined(CRTSCTS)
00075     options.c_cflag &= ~CRTSCTS;
00076 # endif
00077     // Raw IO
00078     options.c_lflag &= ~(ICANON | ECHO | ECHOE | ISIG);
00079     options.c_oflag &= ~OPOST;
00080
00081     // 5 sec timeout
00082     options.c_cc[VMIN] = 0;
00083     options.c_cc[VTIME] = 50;
00084
00085     // Set the new options for the port...
00086     tcsetattr(fd, TCSANOW, &options);
00087
00088     return fd;
00089 }
00090
```

9.13 Файл lib/serial_win.c

#include "serial.h"

Граф включаемых заголовочных файлов для serial_win.c:



9.13.1 Подробное описание

Serial lib WIN32

Автор:

Vladimir Ermakov

См. определение в файле serial_win.c

9.14 serial_win.c

```
00001 /*
00002  *  libORFA -- PC-side gate libruary for Open Robotics Firmware Architecture
00003  *
00004  *  Copyright (c) 2009 Vladimir Ermakov
00005  *
00006  *  Permission is hereby granted, free of charge, to any person obtaining a copy
00007  *  of this software and associated documentation files (the "Software"), to deal

00008  *  in the Software without restriction, including without limitation the rights
00009  *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  *  copies of the Software, and to permit persons to whom the Software is
00011  *  furnished to do so, subject to the following conditions:
00012  *
00013  *  The above copyright notice and this permission notice shall be included in
00014  *  all copies or substantial portions of the Software.
00015  *
00016 * THE SOFTWARE IS PEOLOVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPEOLESS O
      R
00017  *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 * FITNESS FOR A PSWACKRTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL T
      HE
00019  *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

00021  *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00022  *  THE SOFTWARE.
00023  ******************************************************************************/
00029 #include "serial.h"
00030
00031 #define W32SERBUFSIZE 4096
00032
00033 // thanks avrdude
00034 #define FORMATMESSAGE(lpMsgBuf) \
00035         LPVOID lpMsgBuf; \
00036         FormatMessage( \
00037                 FORMAT_MESSAGE_ALLOCATE_BUFFER | \
00038                 FORMAT_MESSAGE_FROM_SYSTEM | \
00039                 FORMAT_MESSAGE_IGNORE_INSERTS, \
00040                 NULL, \
00041                 GetLastError(), \
00042                 /* default language */ \
00043                 MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), \
00044                 (LPTSTR) &lpMsgBuf, \
00045                 0, NULL);
00046
00047
00048 HANDLE open_and_configure(char *port, int baudrate)
00049 {
00050     COMMCONFIG comm_config;
00051     COMMTIMEOUTS comm_timeouts;
00052     DWORD comm_cfg_size = sizeof(COMMCONFIG);
00053     comm_config.dwSize = comm_cfg_size;
00054
00055     HANDLE fd = CreateFileA(port, GENERIC_READ | GENERIC_WRITE, 0, NULL,
00056             OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL, NULL);
00057     if (fd == INVALID_HANDLE_VALUE) {
00058         FORMATMESSAGE(lpMsgBuf);
00059         fprintf(stderr, "open_and_configure: can't open device \"%s\": %s\n",
00060                 port, (char*)lpMsgBuf);
00061         LocalFree(lpMsgBuf);
00062
00063         return INVALID_HANDLE_VALUE;
00064     }
00065
00066     if (!SetupComm(fd, W32SERBUFSIZE, W32SERBUFSIZE)) {
```

```
00067         CloseHandle(fd);
00068         fprintf(stderr, "open_and_configure: can't set buffers for \"%s\"\n",
00069                 port);
00070
00071         return INVALID_HANDLE_VALUE;
00072     }
00073
00074     // configure port settings
00075     GetCommConfig(fd, &comm_config, &comm_cfg_size);
00076     GetCommState(fd, &(comm_config.dcb));
00077
00078     // set baudrate
00079     switch (baudrate) {
00080         case 9600:   comm_config.dcb.BaudRate = CBR_9600;   break;
00081         case 115200: comm_config.dcb.BaudRate = CBR_115200; break;
00082         default:
00083             fprintf(stderr, "Unsupported baudrate %i, set to 9600\n",
00084                     baudrate);
00085             comm_config.dcb.BaudRate = CBR_9600;
00086             break;
00087     }
00088
00089     // flow control off
00090     comm_config.dcb.fOutxCtsFlow = FALSE;
00091     comm_config.dcb.fDtrControl = DTR_CONTROL_DISABLE;
00092     comm_config.dcb.fRtsControl = RTS_CONTROL_DISABLE;
00093     comm_config.dcb.fInX = FALSE;
00094     comm_config.dcb.fOutX = FALSE;
00095
00096     // Binary, 8N1
00097     comm_config.dcb.fBinary = TRUE;
00098     comm_config.dcb.fAbortOnError = FALSE;
00099     comm_config.dcb.fNull = FALSE;
00100     comm_config.dcb.ByteSize = 8;
00101     comm_config.dcb.fParity = NOPARITY;
00102     comm_config.dcb.StopBits = ONESTOPBIT;
00103
00104     // Set the new options...
00105     SetCommConfig(fd, &comm_config, sizeof(COMMCONFIG));
00106
00107     // timeout 5 sec
00108     comm_timeouts.ReadIntervalTimeout = 5000;
00109     comm_timeouts.ReadTotalTimeoutMultiplier = 5000;
00110     comm_timeouts.ReadTotalTimeoutConstant = 0;
00111     comm_timeouts.WriteTotalTimeoutMultiplier = 5000;
00112     comm_timeouts.WriteTotalTimeoutConstant = 0;
00113
00114     SetCommTimeouts(fd, &comm_timeouts);
00115
00116     return fd;
00117 }
00118
00119 int posix_write(HANDLE handle, char *buf, size_t size)
00120 {
00121     int ret;
00122     DWORD written;
00123     if (!WriteFile(handle, buf, size, &written, NULL)) {
00124         FORMATMESSAGE(lpMsgBuf);
00125         fprintf(stderr, "posix_write: write error: %s\n", (char*)lpMsgBuf);
00126         LocalFree(lpMsgBuf);
00127
00128         ret = -1;
00129     } else {
00130         ret = written;
00131     }
00132
00133     return ret;
```

```
00134 }
00135
00136 int posix_read(HANDLE handle, char *buf, size_t count)
00137 {
00138     int ret;
00139     DWORD read;
00140     if (!ReadFile(handle, buf, count, &read, NULL)) {
00141         FORMATMESSAGE(lpMsgBuf);
00142         fprintf(stderr, "posix_read: read error: %s\n", (char*)lpMsgBuf);
00143         LocalFree(lpMsgBuf);
00144
00145         ret = -1;
00146     } else {
00147         ret = read;
00148     }
00149
00150     return ret;
00151 }
00152
```

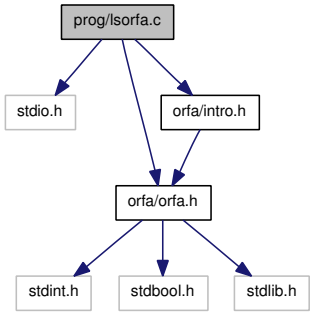
9.15 Файл prog/lsorfa.c

#include <stdio.h>

#include <orfa/orfa.h>

#include <orfa/intro.h>

Граф включаемых заголовочных файлов для lsorfa.c:



9.15.1 Подробное описание

lsorfa Print ORFA drivers table. And it is a simple example how to use liborfa.so

Автор:

Vladimir Ermakov

Необходимо сделать

Add baudrate cli option
Add address cli option

См. определение в файле lsorfa.c

9.16 lsorfa.c

```
00001 /*
00002  *  libORFA -- PC-side gate libruary for Open Robotics Firmware Architecture
00003  *
00004  *  Copyright (c) 2009 Vladimir Ermakov
00005  *
00006  *  Permission is hereby granted, free of charge, to any person obtaining a copy
00007  *  of this software and associated documentation files (the "Software"), to deal

00008  *  in the Software without restriction, including without limitation the rights
00009  *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00010  *  copies of the Software, and to permit persons to whom the Software is
00011  *  furnished to do so, subject to the following conditions:
00012  *
00013  *  The above copyright notice and this permission notice shall be included in
00014  *  all copies or substantial portions of the Software.
00015  *
00016  *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00017  *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00018 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00019  *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00020 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,

00021  *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
00022  *  THE SOFTWARE.
00023  ******************************************************************************/
00035 #include <stdio.h>
00036 #include <orfa/orfa.h>
00037 #include <orfa/intro.h>
00038
00039 void print_intro_table(struct orfadrivers *head)
00040 {
00041     // XXX: check drivers before call this function
00042     struct orfadrivers *drivers=head;
00043
00044     printf("   UID Version Start Count\n");
00045     printf("------ ------- ----- -----\n");
00046
00047     do {
00048         printf("0x%04X %5i.%1i  0x%02X %5i\n", drivers->uid,
00049                 drivers->major_version, drivers->minor_version,
00050                 drivers->start_addr, drivers->count);
00051         drivers = drivers->next;
00052     } while (drivers != NULL);
00053 }
00054
00055 int main(int argc, char *argv[])
00056 {
00057     int ret=0, close_ret;
00058     char *version_str;
00059     struct orfadev *device;
00060     struct orfadrivers *drivers=NULL;
00061
00062     if (argc != 2) {
00063         fprintf(stderr, "Usage: %s /dev/ttyS0\n", argv[0]);
00064         return 1;
00065     }
00066
00067     // TODO: baudrate option
00068     device = orfa_open(argv[1], 115200);
00069     if (device == NULL) {
00070         fprintf(stderr, "Can't open the port: %s\n", argv[1]);
00071         return 1;
00072     }
00073
00074     // try:
```

```
00075     version_str = orfa_get_version(device);
00076     if (version_str == NULL) {
00077         perror("orfa_get_version: return NULL\n");
00078         ret = 2;
00079         goto close_and_exit;
00080     }
00081
00082     // 0 — best choice for local address
00083     if (!orfa_set_local(device, 0)) {
00084         perror("orfa_set_local() fails!\n");
00085         ret = 3;
00086         goto close_and_exit;
00087     }
00088
00089     // TODO: address option
00090     drivers = orfa_get_drivers(device, -1);
00091     if (drivers == NULL) {
00092         perror("orfa_get_drivers() returned NULL\n");
00093         ret = 4;
00094         goto close_and_exit;
00095     }
00096
00097     printf("Protocol: V%s\n\n", version_str);
00098     print_intro_table(drivers);
00099     orfa_free_drivers(&drivers);
00100     free(version_str);
00101
00102     // finally:
00103 close_and_exit:
00104     if ((close_ret = orfa_close(&device))) {
00105         fprintf(stderr, "orfa_close() returned: %i\n", close_ret);
00106         ret = 5;
00107     }
00108
00109     return ret;
00110 }
00111
```

Предметный указатель